

Integrating WebServices with Camel

Daniel Kulp

VP - Open Source Development

02/25/13



Agenda

- Introductions
- What is a Webservice?
- “Low Level” integration
- Full Featured options
- REST

Who Am I

→ Employed by Talend

- VP – Open Source Development
- Team of 8 people devoted to Apache projects
- Working on WebService/SOA related technology for over 10 years

→ Apache Software Foundation

- Involved since Apache CXF entered the incubator in 2006
- PMC Chair of Apache CXF
- Apache Maven, Apache WebServices, Apache Camel, Apache ServiceMix, Apache Aries (11.7K commits)
- Mentored couple other incubator projects
- ASF Member

What is a WebService?

- Exchange of data via standard Web protocols (http/https)
- Primarily associated with SOAP and other WS-* specs
- Include REST
- WebServices in Apache Camel

Low Level Integration

- Bytes come in -> [Transform/Route ->] Bytes out
- Payload agnostic
- Meets the needs of most integration scenarios
- Stream Based

- Two categories of components
 - Consumers/Endpoints
 - Producers

 - Several components are both

Low Level Endpoints

→ camel-servlet

- Provides endpoints for HTTP requests that arrive at a HTTP endpoint that is bound to a published Servlet.

```
<route>  
  <from uri="servlet://foo?serviceName=MyServlet" />  
  ...  
</route>
```

→ camel-jetty

- Provides endpoints for HTTP requests that arrive at a Camel maintained Jetty instance

```
from("jetty:http://localhost:8080/myapp/myservice")  
  .process(new MyProcessor());
```

Low Level Producers

→ camel-http

- Apache HTTP Client 3.1 based

→ camel-http4

- Apache HTTP Client 4.x based

→ camel-ahc

- AsyncHttpClient 1.7.x based

→ camel-jetty

- Eclipse Jetty Client 7.6 based

Low Level Examples

→ Very Simple Proxy Route

```
from("jetty:http://localhost:8080/ProxyService")  
  .to("http://realserver:9000/RealService");
```

→ XSLT Transform

```
<camelContext>  
  <route>  
    <from uri="jetty:http://localhost:8080/myapp/myervice"/>  
    <to uri="xslt:org/apache/camel/spring/processor/example.xsl"/>  
    <to uri="jetty:http://realserver.com:9000/TheURL"/>  
  </route>  
</camelContext>
```


Low Level Examples (cont)

→ XQuery

```
<camelContext>
  <route streamCache="true">
    <from uri="jetty:http://localhost:9000/Service" />
    <choice>
      <when>
        <xquery xmlns:soap="..." xmlns:s="..." >
          /soap:Envelope/soap:Body/s:stocks/symbol='IBM' </xquery>
        <to uri="http://server1/Service" />
      </when>
    </choice>
  </route>
</camelContext>
```

Low Level Components

→ camel-soap

- Data Format which uses JAXB2 and JAX-WS annotations to marshal and unmarshal SOAP payloads.

```
SoapJaxbDataFormat soap = new
    SoapJaxbDataFormat("com.example.customerservice");
soap.setVersion("1.2");

from("direct:start")
    .marshal(soap)
    .to("http://myserver:8080/Service");
```

- CAREFUL - not streaming

Full Featured

→ Spring WebServices (camel-spring-ws)

- Handles the creation of the SOAP wrappers
- Provides some WS-* support
 - WS-Addressing
 - SOME WS-Security Support

→ Producer

```
from("direct:example").to("spring-ws:http://foo.com/bar")
```

→ Consumer

```
from("spring-ws:rootqname:{http://example.com/}GetFoo?  
endpointMapping=#endpointMapping")  
    .convertBodyTo(String.class).to(...)
```

```
from("spring-ws:xpathresult:abc?expression=//  
foobar&endpointMapping=#endpointMapping")  
    .convertBodyTo(String.class).to(...)
```

Full Featured

→ Apache CXF (camel-cxf)

- Full WS-* Support - WS-Addressing, WS-RM, WS-Security, WS-Policy, etc...
- Options for “RAW”, “CXF_MESSAGE”, “PAYLOAD”, and “POJO” models
- JAX-WS Annotation Processing, WSDL Generation
- Complete configuration of CXF features, interceptors, properties, etc...
- HOWEVER - configuration complexity

Camel-CXF

→ CXF Example

```
<cxf:cxfEndpoint id="routerEndpoint"
  address="http://localhost:9003/CamelContext/RouterPort"
  serviceClass="org.apache.hello_world_soap_http.GreeterImpl"/>
```

```
<cxf:cxfEndpoint id="serviceEndpoint"
  address="http://localhost:9000/SoapContext/SoapPort"
  wsdlURL="testutils/hello_world.wsdl"
  serviceClass="org.apache.hello_world_soap_http.Greeter"
  endpointName="s:SoapPort"
  serviceName="s:SOAPService"
  xmlns:s="http://apache.org/hello_world_soap_http" />
```

```
<camelContext>
  <route>
    <from uri="cxf:bean:routerEndpoint" />
    <to uri="cxf:bean:serviceEndpoint" />
  </route>
</camelContext>
```

Camel-CXF DataFormat

→ POJO (default)

- Leverages CXF's DataBindings (JAXB, JIBX, XmlBeans, etc..)

→ RAW/MESSAGE

- The raw stream of bytes from the CXF transport
- VERY little processing done by CXF
- Most interceptors removed

→ PAYLOAD

- Contents of Body as XML Source

→ CXF_MESSAGE

- Full XML Source of the entire message AFTER CXF processing
- WS-Security, WS-RM, etc...
- No streaming right now. :-)

REST

→ camel-restlet

```
from("restlet:http://localhost:8080/users/{id}/basic")
    .process(new Processor() {
        public void process(Exchange exchange)
            throws Exception {
            String id = exchange.getIn()
                .getHeader("id", String.class);
            exchange.getOut().setBody(id + ";Donald Duck");
        }
    });

from("direct:start")
    .to("restlet:http://localhost:8080/users/123/basic")
```

REST

→ camel-cxf using JAX-RS

```
@Path("/customerservice/")
public interface CustomerService {

    @GET
    @Path("/customers/{id}/")
    Customer getCustomer(@PathParam("id") String id);

    @PUT
    @Path("/customers/")
    Response updateCustomer(Customer customer);
}
```

```
from("cxfrs://http://localhost:8080/rest?resourceClasses=...CustomerService")
    .process(new Processor() {
        public void process(Exchange exchange) throws Exception {
            String operationName = inMessage.getHeader(CxfConstants.OPERATION_NAME,
                String.class);

            if ("updateCustomer".equals(operationName)) {
                Customer customer = inMessage.getBody(Customer.class);
                .....
            }
        }
    });
```


If all else fails

→ camel-bean

- Use generated code
- JAX-WS, Axis2, etc...

```
MyService service = new MyService(wsdlUrl,serviceName);  
MyPort proxy = service.getMyPort();  
from("direct:start").bean(proxy, "sayHi");
```

More information

→ Apache Camel

- <http://camel.apache.org>
- users@camel.apache.org
- dev@camel.apache.org

→ Me

- dkulp@apache.org
- dkulp@talend.com
- <http://dankulp.com/blog>